# PIC 마이크로컨트롤러에 USB 플래시 드라이버 인터페이싱 하기

최근 저비용 플래시 메모리 드라이브와 각종 USB 주변기기가 출시됨에 따라 USB 인터페이스는 어디에서든 찾아볼 수 있다. 그러나 이들 주변기기를 8비트 또는 16비트 임베디드 분야에 사용하려고 할 때에는 구현, 비용 및 전력 소모가 중요한 문제가 된다. 이 글에서는 최고 속도의 USB 2.0 인터페이스를 통해 저비용 PIC 마이크로컨트롤러를 플래시 드라이브로 연결하는 방법과 관련 프로그래밍에 대해 설명하고자 한다.

글 : 프레드(Fred Dart), 매니징 디렉터 / Future Technology Devices International (FTDI)
www.ftdichip.com

본고에서는 최고 속도의 USB 2.0 인터페이스를 통해 저비용 PIC 마이크로컨트롤러를 플래시 드라이브로 연결하는 방법과 관련 프로그래밍에 대해 자세히 설명하고자 한다. PIC 마이크로컨트롤러 및 Vinculum USB 인터페이스 칩 주변에 위치한 임베디드 인터페이스의 하드웨어 설계와 더불어 다양한 임베디드 애플리케이션에서 유비쿼터스 USB 플래시 드라이브가 착탈식 스토리지로 사용될 수 있도록 하는 일반적인 프로그래밍에 대해 살펴볼 예정이다.

## 서론

저비용 플래시 메모리 드라이브와 각종 USB 주변기기가 출시됨에 따라 USB 인터페이스는 오늘날 어디에서든 찾아볼 수 있다. 그러나 이들 대부분은 PC 업계에 집중되어 있다. 또한 이들 주변기기를 8비트 또는 16비트 임베디드 분야에 사용하려고 할 때에는 구현, 비용 및 전력 소모가 중요한 문제가 된다. 이러한 문제는 그러한 시스템에 사용되는 임베디드 컨트롤러도 마찬가지이다. 마이크로칩의 PIC 컨트롤러 제품군과 같은 디바이스들은 다양한 메모리 집적도 및 주변기기와 함께 사용되지만, USB 호스트 컨트롤러를 포함시키기 위한 인터페이스, 리소스 및 성능이 부족하다.

이 애플리케이션 예에서 VNC1L Vinculum 컨트롤러 IC는 시스템 컨트롤러로써의 PIC와 USB 2.0 최고 속도 포트 사이에 인터페이스를 제공한다. 이로써, 예를 들어, USB 플래시 메모리 드라이브 연결 시 구현 시간과 오버헤드를 최소화 할 수 있다.

## Vinculum

이 컨트롤러는 데이터 전송을 가속화 하기 위한 쌍둥이 DMA(direct memory access) 엔진과 파일 시스템을 위한 계산을 최적화하기 위한 32비트 수치 코프로세서(numeric co?processor)와 함께 맞춤형 프로세서 주변에 위치해 있다. 이 모든 것은 64Kbyte의 임베디드 플래시 프로그램 메모리와

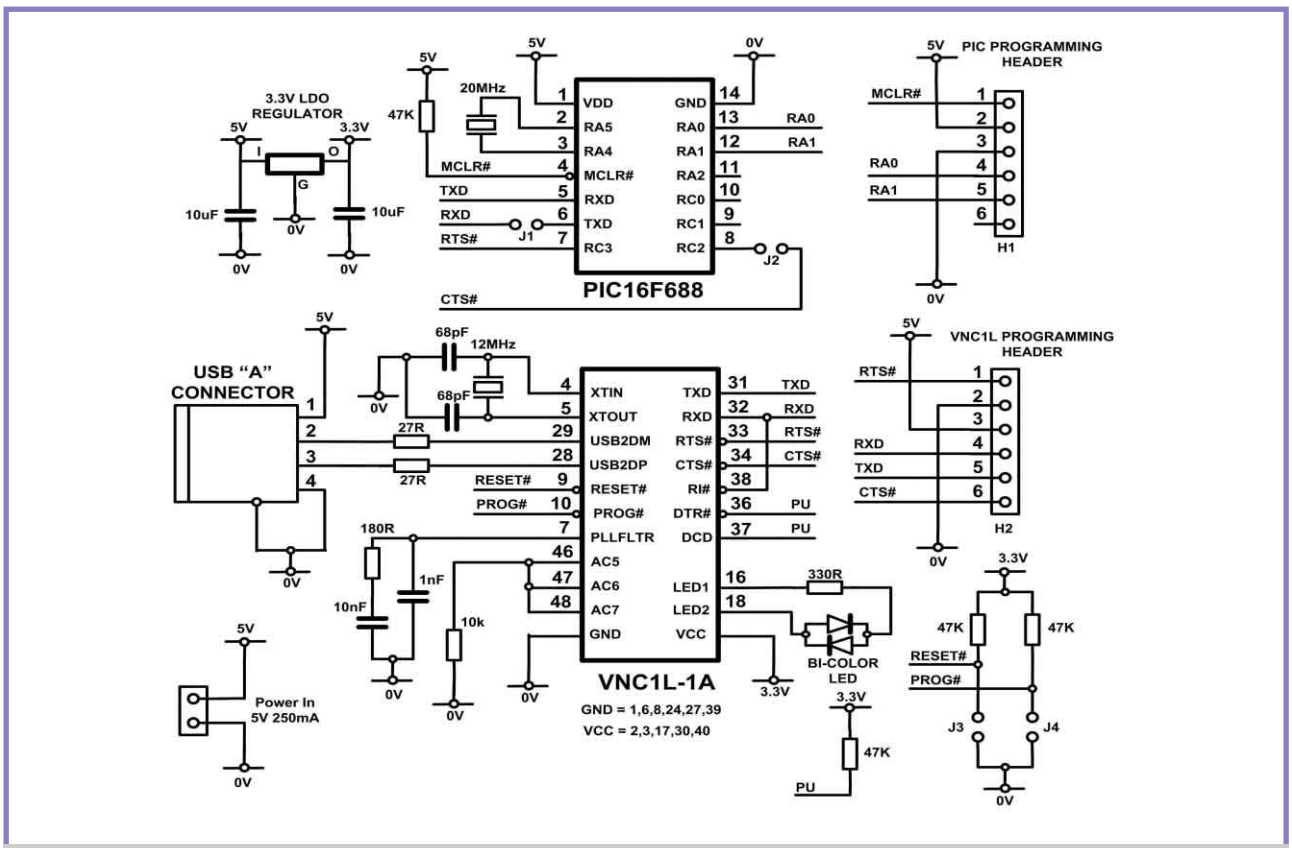


그림 1. Interfacing a USB Flash Drive to a PIC Microcontroller

4Kbyte의 내부 데이터 SRAM을 탑재한 싱글칩에 결합되어 있다. Vinculum은 특별히 임베디드 USB 컨트롤러 시장을 겨냥해 고안되었으며, 최소한의 외부 부품만 있으면 된다. Vinculum 코어의 한 가지 핵심 특징은 일반 MCU 코어와 비교해 코드 길이가 매우 짧아졌다는 것이다. 코어의 코드 오버헤드를 줄임으로써 온칩 e-Flash 메모리에 더 많은 기능을 포함시킬 수 있다. 이와 같은 특징들은 PIC 기반의 임베디드 시스템에 보완적 역할을 한다. 그림 1은 이러한 시스템의 다이어그램을 나타낸 것으로, Vinculum을 사용해 소형 PIC MCU를 USB 'A' 커넥터, 그리고 USB 플래시 드라이브에 연결했다.

## 개요 설명

이 PIC는 시스템의 컨트롤러로, 범용 I/O 핀(핀 9, 10, 11에서 RC0, RC1, RA2)을 통해 센서 또는 다른 소스로부터 데이터를 받아서 플래시 드라이브에 있는 파일로의 스트림에서 데이터 포맷 변환 및 데이터 쓰기를 한다. 명령과 데이터는 TXD(핀 6)을 통해 VNC1L RXD(핀 32)로 보내진다. VNC1L은 FAT 12/16/32 파일 생성과 USB 플래시 드라이브에서의 데이터 스토리지를 담당하며, 핀 28, 29에서 USB2DM 및 USB2DP를 통해 드라이브와 통신한다. 데이터는 같은 핀을 통해 플래시 드라이브로부터 읽혀지

고, PIC의 VNC1L TXD(핀 31)에서 RXD(핀 5)로 보내져 시스템 펌웨어에 의해 사용된다.

이 시스템은 PIC의 펌웨어에 의해 제어되는데, PIC가 내린 명령에 의해 전송이 제어되고, Vinculum의 표준 펌웨어에 의해 해석된다. 이는 시스템을 간단하게 설명한 것으로, 설계를 완성하는 데에는 더 복잡한 것들이 요구된다. 이들 디바이스는 전력, 클록 제어를 위한 크리스털을 필요로 하며, 이들은 프로그래밍 작업도 거쳐야 한다.

PIC의 핀 2, 3에서 20MHz의 크리스털을 사용함으로써 UART 인터페이스에서 최대 115200bps의 높은 변조 속도가 가능해 (내부 8MHz의 오실레이터를 이용하면 최대 9600bps 정도 밖에 얻지 못한다) 시스템의 성능을 향상시킬 수 있다. PIC 펌웨어에 의해 사용되는 PIC IO 핀 RC2 및 RC3은 VNC1Ls UART 인터페이스로 RTS/CTS 핸드세이크 신호(handshake signal)를 시뮬레이트 한다.

250mA에서 5V의 레귤레이트 된 PSU가 필요하며, USB 'A' 커넥터에서 최대 200mA를 제공한다. VNC1L에 전력을 공급하는 데에는 25mA, PIC16F688은 25mA의 전력이 필요하다. VNC1L은 3.3V의 공급 전압을 필요로 하는데, 이는 3.3V LDO 레귤레이터에서 제공되며, 레벨 시프터를 사용하지 않고 PIC로 연결될 수 있도록 하는 5V의 허용오차 I/O핀을 가지고 있다.

저전력 애플리케이션을 위해 VNC1L은 필요 없을 경우 2mA의 슬립 모드로 설정될 수 있다. 다시 디바이스를 활성화시키기 위해서는 UART 인터페이스의 링 인디케이터(RI) 핀을 스트로브(strobe) 한다. 여기에서처럼, 이것이 RXD 라인에 연결되면, 들어오는 더미 명령(dummy command)에 의해 트리거링 되어 디바이스를 활성화시킬 수 있다.

이 디자인은 또한 핀 16 및 18로부터 전력을 공급받는 2가지색 LED 인디케이터를 포함하고 있다. 이는 USB 드라이브의 성공적인 열거(enumeration)과 파일 시스템으로의 액세스를 나타내준다.

### VNC1L 펌웨어

ViNC1L은 VDAP(Vinculum Disk and Peripheral)라는 표준 펌웨어로 프로그래밍되며, 이는 PIC로부터의 명령을 해석한다. 이들 VDAP 명령은 DIR, RD 및 WR 등 DOS와 같은 명령이다. 이 명령셋은 또한 마이크로프로세서 제어 방식에 가장 적합한 싱글 바이트 헥스(hex) 명령어를 지원한다.

VDAP 명령은 PIC 펌웨어에 포함되어 있으며, USB 플래시 드라이브로의 액세스를 제어한다. 일반적인 시퀀스는 파일을 생성하고, 파일로의 데이터 읽기/쓰기 및 파일 닫기에 사용된다.

## VNC1L 및 PIC 프로그래밍

개발 환경이 요구된다고 가정하여 이 설계는 각각의 디바이스를 위한 2개의 프로그래밍 헤더(header)를 포함하고 있다. 생산 설계를 위해 두 디바이스 모두 PCB에서 삽입 전에 미리 프로그래밍 될 수 있어 헤더와 점퍼(jumper)를 제거한다.

일반 동작 동안 J1 및 J2는 파퓰레이트(populate) 되어야 하고, 다른 점퍼들은 오픈 상태로 있어야 한다. VNC1L을 프로그래밍하기 위해 J1과 J2 점퍼를 제거해 VNC1L UART 입력을 PIC 출력으로부터 절연시킨다. 5V PSU를 차단하고 TTL-232R-3V3 케이블을 H2에 연결시킨다. 이 케이블의 USB 사이드를 VPROG 프로그래밍 유틸리티가 설치된 PC로 연결시킨다. VNCL1A의 PROG# 핀을 낮게 당기기 위해 J4를 파퓰레이트 한 후 디바이스를 리셋하고 프로그래밍 모드로 들어가게 하기 위해 일시적으로 J3을 단락시킨다. 프로그래밍 이후에는 점퍼 설정을 동작 모드로 재저장 하는 것을 잊지 말아야 한다.

PIC용 프로그래밍 헤더는 디바이스의 핀 RA0 및 RA1과 MCLR#에 연결되고, 헤더를 통해 5V 프로그래밍 전압/서플라이가 공급된다. PIC 마이크로컨트롤러 프로그래밍 이전에 5V PSU를 차단해야 한다. PICKit2와 같은 표준 PIC 개발 환경에 헤더를 연결해 마이크로칩의 디버그 및 다운로딩 툴을 사용할 수도 있다.

PIC가 제공하는 샘플 C 코드는 플래시 디스크가 감지되고 hello.txt라는 파일을 열 때까지 기다린다. 그 다음 캐리지 리턴

(carriage return) 및 라인 피드(line feed) 캐릭터와 함께 텍스트 'Hello World' 가 파일로 쓰여진다. 그리고 나서 파일을 닫고, 디스크가 제거될 때까지 기다린다.

## 결론

FTDI의 Vinculum VNC1L은 사용하기 쉬우며, 저비용 마이크로컨트롤러와 USB 2.0 저/고속 주변기기 간의 인터페이스 프로그래밍도 쉽게 할 수 있다. DOS와 같은 명령셋을 통해 마이크로컨트롤러 환경 내에서 데이터 전송 루틴을 쓰고, 디버깅을 쉽게 할 수 있다. 단순한 레이아웃 또한 임베디드 시스템을 위한 저비용 USB 호스트 구현이 가능하도록 해준다. 이로써 저비용, 유비쿼터스 USB 플래시 드라이브를 시스템을 위한 데이터 스토리지 미디어로 사용할 수 있을 뿐만 아니라 필드에서 소프트웨어 업그레이드 제공이 가능하다. 이 글의 주제에서 약간 벗어나긴 하지만, VNC1L 디바이스는 또한 매스 스토리지 디바이스 외에 다른 많은 USB 주변기기를 연결하는데도 사용될 수 있다.

Vinculum IC는 또한 USB 인터페이스의 전력 예산의 10% 정도만 추가하면 되며, 시스템 전력 예산에는 이보다 훨씬 적게 든다. USB2.0 호스트 컨트롤러는 휴대용 디바이스에 쉽고 간단하게 추가될 수 있다.

VDAP 펌웨어와 완전한 명령셋이 기술된 문서는 FTDI의 Vinculum 웹사이트(http://www.vinculum.com)에서 다운로드 할 수 있다.

### 소스 코드

```
//*********************************
//
// Code for logging data directly to a USB Flash disk
// using a PIC16F688 and the VNC1L.
// Commands received through UART.
// The PIC only handles Tx and Rx automatically
// for the UART, so handshaking must
// be implemented in code.
//
// Author: FTDI
//
// Code written for PIC16F688 using SourceBoost C compiler
// (www.sourceboost.com)
//
//*********************************

//*********************************
//
// Pin definitions for PIC16F688 on VPIC board
//
// PIC16F688 Pin Number Designation Function
// 1                VDD      Power
// 2                RA5      OSC_IN
// 3                RA4      OSC_OUT
// 4                MCLR#    MCLR#
// 5                RC5      Rx (UART)
// 6                RC4      Tx (UART)
// 7                RC3      CTS# (connects to RTS#
on TTL cable)
// 8                RC2      RTS# (connects to CTS#
on TTL cable)
// 9                RC1      Analogue/Digital Input
// 10               RC0      Analogue/Digital Input
// 11               RA2      Analogue/Digital Input
(Stop button)
// 12               RA1      Analogue/Digital Input
```

```
(Start button)
// 13                RA          Analogue/Digital Input
(Use this one for analog input)
// 14                GND         Ground
//
//*********************************
#include <BoostCPic16.h>
#include <boostc.h>

#pragma DATA 0x2007, _HS_OSC & _WDT_OFF &
_PWRTE_OFF & _MCLRE_OFF & _CP_OFF & _IESO_OFF &
_FCMEN_OFF;
#pragma CLOCK_FREQ 20000000;

//*********************************
//
// Constants and variables for serial port
//
//*********************************
// Define serial port flags
char _RTS = 0x04      // RTS# is RC2 (Port C bit 2)
char _CTS = 0x08;     // CTS# is RC3 (Port C bit 3)
char FLOW_NONE = 0x00;
char FLOW_RTS_CTS = 0x01;
char FLOW_CONTROL_ENABLED = 0x00;       // Initialise
FLOW_ CONTROL_ENABLED to off

// Define serial port Rx buffer variables
char SERIAL_RX_FIFO[0x20];              // Allocate
buffer for RX FIFO USE MULTIPLE OF 0x04!
char RX_FIFO_SIZE = sizeof(SERIAL_RX_FIFO);   // Initialise
FIFO size variable
char RX_FIFO_COUNT = 0x00;  // Reset RX FIFO count
char * RX_FIFO_HEAD_PTR = &SERIAL_RX_FIFO[0x00];   //
Initialise head pointer
char * RX_FIFO_TAIL_PTR = &SERIAL_RX_FIFO[0x00];   //
Initialise tail pointer
char * RX_FIFO_START = RX_FIFO_TAIL_PTR;   // Set lowest
address of RX_FIFO
char * RX_FIFO_END = RX_FIFO_START + RX_FIFO_SIZE; //
Set highest address of RX_FIFO
char RX_STATUS = 0x00;  // Reset RX status
char RX_FIFO_OVERRUN = 0x01;

char RX_FIFO_UPPER_LIMIT = 0x03 *(RX_FIFO_SIZE / 0x04);
// Buffer limit for handshaking
char RX_FIFO_LOWER_LIMIT = (RX_FIFO_SIZE / 0x04);   //
Buffer limit for handshaking

//*********************************
//
// Define states for state machine
//
//*********************************

enum {              // Enum states
        VNC1L_Idle,         // Idle state
        VNC1L_DiskAvailable,
        VNC1L_CreateFile,
        VNC1L_WriteFile,
        VNC1L_EndFile,
        VNC1L_WaitForRemove
};
char VNC1L_State = VNC1L_Idle;     // Initialise state to idle
```

```
char Synchronised = 0x01;
char GotDisk = 0x01;

////////////////////////////////////////////////////////////////////////
//
//  Routines for serial port
//
////////////////////////////////////////////////////////////////////////

//*********************************
//
//  Reset Rx FIFO
//
//*********************************

void serial_resetrxfifo()
{
        pie1.RCIE = 0x00;           // Disable interrupts
        intcon.GIE = 0x00;
        intcon.PEIE = 0x00;
        RX_FIFO_COUNT = 0x00;       // Reset FIFO count to 0
        RX_STATUS = 0x00;           // Reset FIFO status
        RX_FIFO_HEAD_PTR = (&SERIAL_RX_FIFO[0x00]); //
Set FIFO head pointer to address of first byte of SERIAL_RX_FIFO
        RX_FIFO_TAIL_PTR = (&SERIAL_RX_FIFO[0x00]); //
Set FIFO tail pointer to address of first byte of SERIAL_RX_FIFO
        pie1.RCIE = 0x01;  // Enable interrupts for Rx byte
        intcon.GIE = 0x01;
        intcon.PEIE = 0x01;
        if (FLOW_CONTROL_ENABLED == 0x01)   // If using
flow control
        {
                portc &= ~_RTS;  // Set RTS# active (low)
        }
}

//*********************************
//
//  Initialise serial port (Baud rate, handshaking, interrupt
enable, enable receiver)
//
//*********************************

void serial_init(long BaudRate, char Handshaking)        / /
Set up serial port
{
        pie1.RCIE = 0x00;  // Disable interrupts
        intcon.GIE = 0x00;
        intcon.PEIE = 0x00;

        baudctl = 0x00;
        ansel &= 0x3F;     // Digital IO on RC2, RC3
        trisc = 0xDB;      // Set RC4 (Tx) & RC2 (RTS#) as
output,RC5 (Rx) & RC3 (CTS#) as input
                           // TRISC bits 1 for input, 0 for
output
        portc |= _RTS;     // Set RTS# high (inactive) at
this stage
        txsta = 0x04;      // Enable UART module and set
8,n,1 - Set BRGH to 1
        rcsta.SPEN = 0x01; // Enable serial port

        switch (BaudRate)  // Set Baud rate
```

```
{
        case 9600:
                spbrg = 0x81;
                break;
        case 19200:
                spbrg = 0x40;
                break;
        case 57600:
                spbrg = 0x15;
                break;
        case 115200:
                spbrg = 0x0A;
                break;
        default:
                spbrg = 0x81;   // Default to
9600 if not a valid achievable Baud rate
}
spbrgh = 0x00;


if (Handshaking == FLOW_RTS_CTS)
{
        portc &= ~_RTS;   // If handshaking is
required, set RTS# active (low)
        FLOW_CONTROL_ENABLED = 0x01;
// Set flow control enabled flag
}

serial_resetrxfifo();  // Reset Rx FIFO

pir1.RCIF = 0x00;   // Clear Rx interrupt flag
pie1.RCIE = 0x01;   // Enable interrupts for Rx byte
intcon.GIE = 0x01;
intcon.PEIE = 0x01;
rcsta.CREN = 0x01;  // Enable receiver
}

//*********************************
//
//  Send a byte directly from the transmit register
//
//*********************************

void serial_sendbyte(char TX_BYTE)
{
        if (FLOW_CONTROL_ENABLED == 0x01)
        {
                while (portc & _CTS);        // Wait for
CTS# to become active (low)
        }
        while (!txsta.TRMT);  // Wait for TSR to be empty
        txreg = TX_BYTE;   // Move byte to send into
TXREG
        txsta.TXEN = 1;      // Start transmission
        while (!txsta.TRMT);// Wait for TSR to be empty -
indicates sending is complete
}

//*********************************
//
//  Read a byte directly from the receive character register
//
//*********************************
void serial_receivebyte(char RX_BYTE)
```

```
{
        RX_BYTE = rcreg;
        pir1.RCIF = 0x00;   // Clear Rx interrupt flag
}

//*********************************
//
//  Add a byte from the receive character
//  register to the receive FIFO buffer
//
//*********************************

void serial_addtorxfifo()
{
        if (RX_FIFO_COUNT >= RX_FIFO_SIZE)   // If Rx FIFO is full
        {
                RX_STATUS = RX_STATUS | RX_FIFO_OVERRUN;        // Set overrun flag
                if (FLOW_CONTROL_ENABLED == 0x01)/  / If using flow control
                {
                        portc |= _RTS;   // Set RTS# inactive (high)
                }
                return;
        }

        if (RX_FIFO_COUNT >= RX_FIFO_UPPER_LIMIT)   // Keep some space in the FIFO for bytes already being sent
        {
                if (FLOW_CONTROL_ENABLED == 0x01)/  / If using flow control
                        {
                                portc |= _RTS;
// Set RTS# inactive (high)
                        }
        }

        intcon.GIE = 0x00; // Disable interrupts
        intcon.PEIE = 0x00;

        *RX_FIFO_HEAD_PTR++ = rcreg;   //Add received byte to RX_FIFO and then increment head pointer
        RX_FIFO_COUNT++;// Increment byte count

        if (RX_FIFO_HEAD_PTR >= RX_FIFO_END)   // If head pointer is greater than allocated FIFO end address
        {
                RX_FIFO_HEAD_PTR = RX_FIFO_START; /  / Wrap-around buffer to start address again (circuluar FIFO)
        }

        if ((RX_FIFO_COUNT < RX_FIFO_UPPER_LIMIT) && (FLOW_CONTROL_ENABLED == 0x01))  // If Rx FIFO is not full
        {
                portc &= ~_RTS;  // Set RTS# active (low)
        }

        intcon.GIE = 0x01;  // Re-enable interrupts
        intcon.PEIE = 0x01;
}
//*********************************
//
```

```
// Read a byte out of the receive FIFO buffer
//
//*********************************

void serial_readfromrxfifo(char *RX_FIFO_BYTE)
{
        if (RX_FIFO_COUNT == 0)  // If FIFO is empty
        {
                return;   // Exit - no data to read
        }

        intcon.GIE = 0x00; // Disable interrupts
        intcon.PEIE = 0x00;

        *RX_FIFO_BYTE = *RX_FIFO_TAIL_PTR++;  // Read a
byte from the FIFO at position given by RX_FIFO_TAIL_PTR
and then increment tail pointer
        RX_FIFO_COUNT--;// Decrement byte count

        if (RX_FIFO_TAIL_PTR >= RX_FIFO_END)   // If tail
pointer is greater than allocated FIFO end address
        {
                RX_FIFO_TAIL_PTR = RX_FIFO_START;   //
Wrap-around buffer to start address again (circuluar FIFO)
        }

        intcon.GIE = 0x01;  // Re-enable interrupts
        intcon.PEIE = 0x01;

        if ((RX_FIFO_COUNT < RX_FIFO_LOWER_LIMIT) &&
(FLOW_CONTROL_ENABLED == 0x01))  // If using flow control
        {
                portc &= ~_RTS;   // Set RTS# active
(low)
        }
}

////////////////////////////////////////////////////////////////////////
//
//  VNC1L VDAP commands go here
//
////////////////////////////////////////////////////////////////////////

//*********************************
//
//  Echo command using E - same for short command set or
//  long command set
//
//*********************************

void VNC1L_Big_E()
{
        serial_sendbyte('E');// Send 'E'
        serial_sendbyte(0x0D);      //  Send  carriage
return
}

//*********************************
//
//  Echo command using e - same for short command set or
//  long command set
//
//*********************************
void VNC1L_Small_e()
```

```
{
        serial_sendbyte('e');// Send 'e'
        serial_sendbyte(0x0D);      //  Send  carriage
return
}

//*********************************
//
//  Synchronise to the VNC1L using the e and E echo
commands
//
//*********************************

char VNC1L_Sync()
{
        char LoopCount = 0x00;
        char FIFOByte1 = 0x00;
        char FIFOByte2 = 0x00;
        char Got_Big_E = 0x00;
        char Got_Small_e = 0x00;

        serial_resetrxfifo();  //  Reset  FIFO  before
synchronising

        while ((Got_Big_E == 0x00) && (LoopCount < 0xFF))
        {
                VNC1L_Big_E();              // Send Big
E echo
                LoopCount++;                // Increment
loop count
                while (RX_FIFO_COUNT < 0x02)
                {
                        delay_ms(100);              / /
Wait for 2 byte response
                }

                if (RX_FIFO_COUNT > 0x00)  // If data is
available
                {
                        while (RX_FIFO_COUNT > 0x02)
// Read all data available except last 2 bytes
                        {
serial_readfromrxfifo(&FIFOByte1);      // If more than 2 bytes
available, read surplus ones
                        }
serial_readfromrxfifo(&FIFOByte1);      //  Check  that
remaining 2 bytes are 'E' and 0x0D
serial_readfromrxfifo(&FIFOByte2);
                        if  ((FIFOByte1  ==  'E')  &&
(FIFOByte2 == 0x0D))
                        {
                                Got_Big_E = 0x01;
                        }
                        else
                        {
                                delay_ms(10);       / /
Wait a bit and retry synchronisation
                        }
                }
```

```
        }

        if (Got_Big_E == 0x01)
        {
                VNC1L_Small_e();
                while (RX_FIFO_COUNT < 0x02)
                {
                        delay_ms(10);       // Wait for 2
byte response
                }
                serial_readfromrxfifo(&FIFOByte1);      //
Check that remaining 2 bytes are 'e' and 0x0D
                serial_readfromrxfifo(&FIFOByte2);
                if ((FIFOByte1 == 'e') && (FIFOByte2 ==
0x0D))
                {
                        Got_Small_e = 0x01;         //
If small e found, then synchronised
                }

        }

        return Got_Small_e;

}

//*********************************
//
//  Look for disk - assumes long command set being used
//
//*********************************
char VNC1L_FindDisk()       // Return 0x01 if disk is found,
else return 0x00
{
        char FIFOByte1 = 0x00;
        char FIFOByte2 = 0x00;
        char FIFOByte3 = 0x00;
        char FIFOByte4 = 0x00;
        char FIFOByte5 = 0x00;

        serial_sendbyte(0x0D);              //      Send
carriage return
        while (RX_FIFO_COUNT < 0x05);               //
Wait until at least 5 bytes in the Rx FIFO
        serial_readfromrxfifo(&FIFOByte1);      //      Read
bytes out of Rx FIFO
        serial_readfromrxfifo(&FIFOByte2);
        serial_readfromrxfifo(&FIFOByte3);
        serial_readfromrxfifo(&FIFOByte4);
        serial_readfromrxfifo(&FIFOByte5);

        if ((FIFOByte1 == 'D') && (FIFOByte2 == ':') &&
(FIFOByte3 == 0x5C) && (FIFOByte4 == '>') && (FIFOByte5 ==
0x0D)) // Check for prompt
        {
                return 0x01;        // If prompt found,
then return disk available
        }
        else
        {
                while (RX_FIFO_COUNT > 0x00)
                {
```

```
serial_readfromrxfifo(&FIFOByte1);      // Read any additional
bytes out of the FIFO
                }
                return 0x00;            // Return no disk
available
        }

}

//*********************************
//
//  Open file for write command (OPW) using long command set
//
//*********************************

void VNC1L_OpenFile()
{
        char FIFOByte = 0x00;

        serial_sendbyte('O');           // Send 'O'
        serial_sendbyte('P');           // Send 'P'
        serial_sendbyte('W');           // Send 'W'
        serial_sendbyte(' ');           // Send ' '
        serial_sendbyte('h');           // Send 'h'
        serial_sendbyte('e');           // Send 'e'
        serial_sendbyte('l');           // Send 'l'
        serial_sendbyte('l');           // Send 'l'
        serial_sendbyte('o');           // Send 'o'
        serial_sendbyte('.');           // Send '.'
        serial_sendbyte('t');           // Send 't'
        serial_sendbyte('x');           // Send 'x'
        serial_sendbyte('t');           // Send 't'
        serial_sendbyte(0x0D);          // Send carriage
return
        while (RX_FIFO_COUNT < 0x05)    // Wait until at least 5
bytes in the Rx FIFO
        {
                delay_ms(10); // Wait for 5 byte response("D:\"
+ 0x0D)
        }
        while (RX_FIFO_COUNT > 0x00)
        {
                serial_readfromrxfifo(&FIFOByte);       // /
Read response bytes form FIFO
        }
}

//*********************************
//
//  Write to file command (OPW) using long command set
//
//*********************************

void VNC1L_WriteToFile()
{
        char FIFOByte = 0x00;

        serial_sendbyte('W');           // Send 'W'
        serial_sendbyte('R');           // Send 'R'
        serial_sendbyte('F');           // Send 'F'
        serial_sendbyte(' ');           // Send ' '
        serial_sendbyte(0x00);          // Send 0x00 -
Number of bytes to write MSB
        serial_sendbyte(0x00);          // Send 0x00
```

```
        serial_sendbyte(0x00);      // Send 0x00
        serial_sendbyte(0x0E);      // Send 0x0E - Number of bytes to write LSB
        serial_sendbyte(0x0D);      // Send carriage return
        serial_sendbyte('H');       // Send 'H'
        serial_sendbyte('e');       // Send 'e'
        serial_sendbyte('l');       // Send 'l'
        serial_sendbyte('l');       // Send 'l'
        serial_sendbyte('o');       // Send 'o'
        serial_sendbyte(' ');       // Send ' '
        serial_sendbyte('W');       // Send 'W'
        serial_sendbyte('o');       // Send 'o'
        serial_sendbyte('r');       // Send 'r'
        serial_sendbyte('l');       // Send 'l'
        serial_sendbyte('d');       // Send 'd'
        serial_sendbyte('!');       // Send '1'
        serial_sendbyte(0x0D);      // Send carriage return
        serial_sendbyte(0x0A);      // Send line feed
        serial_sendbyte(0x0D);      // Send carriage return

        while (RX_FIFO_COUNT < 0x05)  // Wait until at least 5 bytes in the Rx FIFO
        {
                delay_ms(10);   // Wait for 5 byte response("D:\>" + 0x0D)
        }

        while (RX_FIFO_COUNT > 0x00)
        {
                serial_readfromrxfifo(&FIFOByte);     // Read response bytes form FIFO
        }

}

//*********************************
//
//  Close file command (CLF) using long command set
//
//*********************************

void VNC1L_CloseFile()
{
        char FIFOByte = 0x00;

        serial_sendbyte('C');       // Send 'C'
        serial_sendbyte('L');       // Send 'L'
        serial_sendbyte('F');       // Send 'F'
        serial_sendbyte(' ');       // Send ' '
        serial_sendbyte('h');       // Send 'h'
        serial_sendbyte('e');       // Send 'e'
        serial_sendbyte('l');       // Send 'l'
        serial_sendbyte('l');       // Send 'l'
        serial_sendbyte('o');       // Send 'o'
        serial_sendbyte('.');       // Send '.'
        serial_sendbyte('t');       // Send 't'
        serial_sendbyte('x');       // Send 'x'
        serial_sendbyte('t');       // Send 't'
        serial_sendbyte(0x0D);      // Send carriage return
```

```
        while (RX_FIFO_COUNT < 0x05)   // Wait until at least 5 bytes in the Rx FIFO
        {
                delay_ms(10);   // Wait for 5 byte response("D:\>" + 0x0D)
        }

        while (RX_FIFO_COUNT > 0x00)
        {
                serial_readfromrxfifo(&FIFOByte);       // Read response bytes form FIFO
        }

}

//*********************************
//*********************************
//
// Interrupt routines go here
//  Use receive character interrupt to add received characters
// to the RX FIFO
//
//*********************************
//*********************************

void interrupt()
{

        if (pir1.RCIF)
        {
                serial_addtorxfifo(); // Add received byte to Rx FIFO
                pir1.RCIF = 0x00;   // Clear RCIF flag
        }
}

//*********************************
//*********************************
//
// Main line code
//
//*********************************
//*********************************

void main()
{
        char Sync = 0x00;

        VNC1L_State = VNC1L_Idle;   // Initialise state to idle
        serial_init(9600, FLOW_RTS_CTS);       // Initialise Baud rate and flow control
        delay_s(1);                            // Allow some time for power up
        if (VNC1L_Sync() == Synchronised)   // Synchronise to Vinculum
        {
                while (1)  // Main state machine - do not return from here
                {
                        switch (VNC1L_State)
                        {
```

```
                    case VNC1L_Idle:
                        Sync = VNC1L_Sync();
                        if (VNC1L_FindDisk() == GotDisk)  // Check for disk
                        {
                            VNC1L_State = VNC1L_DiskAvailable;  // Update state to indicate disk found
                        }
                        break;

                    case VNC1L_DiskAvailable:
                        VNC1L_OpenFile();  // Send open file for write command (OPW) - file name "hello.txt"
                        VNC1L_State = VNC1L_WriteFile; // Update state to indicate file has been created and can be written
                        break;

                    case VNC1L_WriteFile:
                        VNC1L_WriteToFile();  // Send write file command (WRF) - write "Hello World!"
                        VNC1L_State = VNC1L_EndFile;  // Update state to indicate file has been written and can now be closed
                        break;

                    case VNC1L_EndFile:
                        VNC1L_CloseFile();  // Send close file command (CLF)
                        VNC1L_State = VNC1L_WaitForRemove;  // Update state to indicate witing for disk to be removed
                        break;

                    case VNC1L_WaitForRemove:
                        Sync = VNC1L_Sync();
                        if (VNC1L_FindDisk() != GotDisk)  // Check for disk
                        {
                            VNC1L_State = VNC1L_Idle;  // Update state to indicate disk has been removed and return to idle
                        }
                        break;

                    default:
                        VNC1L_State = VNC1L_Idle;  / Default to idle state
                    }
                }
            }
}
```